TCP短期デッドロック問題の解決

村山公保† 山口 英†

TCPはインターネットで最も重要なトランスポート・プロトコルであり，性能の向上が強く求められている．本論文では，TCPの重大な問題であるTCPの短期的なデッドロック問題について議論する．この問題は，TCPの確認応答処理に不備があるために発生する．この問題が発生すると，セグメントが間欠的にしか転送されなくなり，スループットが極端に悪化する．このデッドロック問題の解決法として，確認応答の遅延処理を無効にする方法が提案されているが，デッドロックが起きない場合でも確認応答数が2倍に増加するという問題がある．本論文では，TCPのデータ転送モデルを重視し，送信ホストが受信ホストの遅延確認応答処理を完全に制御することで，確認応答数の増加なしにデッドロックを解決できる方法を提案する．

A Solution of the TCP Short-term Deadlock Problem

YUKIO MURAYAMA† and SUGURU YAMAGUCHI†

Since the TCP is the most important transport protocol in the Internet, it should be strongly encouraged to improve its performance and usability. In this paper, we discuss one of the major drawbacks called "TCP short-term deadlock problem" in the current TCP implementations. Because of inadequate delayed acknowledgment handling, the short-term deadlocks where data are transferred intermittently occur repeatedly under some conditions. As its result, the end-to-end throughput on the TCP connection is extremely decreased. Some solutions have been proposed for this problem, however, they turn off its delayed acknowledgment mechanism in order to eliminate the short-term deadlocks, hence, they add twice acknowledgments. We propose yet another solution for this problem. TCP deadlocks can be eliminated with our method and this solution adds little traffic more, because the TCP sender can control the receiver's processing of the delayed ACK perfectly.

1. はじめに

インターネットの発達とともに，インターネットのプロトコルの中に潜在していた問題が浮き彫りになってきた．たとえば，IPは，今日のような全世界的な規模で運用されるネットワーク上で利用されることを想定した設計がされていなかった．そのため，IPアドレスが枯渇するという問題が発生した．この問題に対処するため，IPの次のバージョンである，IPv6に関する研究・標準化作業が進められている[1]．IPv6は当面の問題であるIPアドレスの枯渇問題に対する解決だけではなく，今までインターネットを運用してきた経験と反省をふまえて，IPプロトコルの不満を一掃できるような仕様が標準化されようとしている．具体的にはルータの処理を簡略化する工夫や，プラグ&プレイ機能，セキュリティ，品質制御（QoS）などの機能を，オプションではなく標準で実装しなければならないことが決められている．オプション機能は必ずしも実装されないことがあり，それがインターネットを運用していくうえで，しばしば問題を引き起こしたからである[2]．

TCPにも問題があることが分かっている．その中のいくつかの問題は，オプションで解決策が提案されている[3),4]．しかし，これらはIPv6上でも特に必須の機能とは定義されておらず，IPv4とまったく同一のTCPが利用されようとしている．これではIPv6上でもTCPの問題点がそのまま残ることになる．IPが改良されたとしても，TCPが同じであれば，現在のインターネットで見られるパフォーマンスの問題は，必ずしも改善されるとは限らない．このため，現在のTCPをできる限り改善し，その結果を標準に反映させる活動を積極的に行う必要がある．

本論文では，TCPの性能に関する問題の中で，副作用のない効果的な解決策が提案されていない，TCP短期デッドロック問題の解決法を提案し，実装して，

† 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology

評価する．これは，TCP によるデータ転送を著しく悪化させる問題であり，ごく一般的な環境でも発生するため，解決しなければならない課題である．

本論文は次のように構成される．まず最初に，TCP の短期デッドロック問題の概要について説明する．次に，これまでに提案されてきた解決案を示し，その問題点を TCP のデータ転送モデルを考慮しながら述べる．そして，本論文で提案する TCP の解決案を示し，実装して検証実験を行った結果について報告する．最後に，本提案のインターネットへの適用について考察する．

2. TCP の短期的なデッドロックの問題

TCP には，Nagle アルゴリズム[5]とスロー・スタート[6]というデータの送信を抑制する機構と，遅延確認応答（Delayed ACK）[7]という確認応答を遅延させる機構が備えられている．これらの処理がうまく噛み合わなくなると，ある期間，データの転送が停止することがある．具体的には次の状態になる．本論文ではこの状態になることを，TCP デッドロックと呼ぶ．

送信ホスト：続けて送信すべきデータがあるが，前に送信したセグメントに対する確認応答を受信するまで，データを送信しない

受信ホスト：確認応答していない受信セグメントがあるが，さらにデータを受信しないと確認応答しない

この状態は，遅延確認応答処理で確認応答が遅延される期間続く．この上限は 0.5 秒に決められている[8]．BSD の実装を基にした多くの実装では，確認応答の遅延時間は最大 0.2 秒間であり，デッドロックは最大 0.2 秒続く．このデッドロックは繰り返し発生することがある．この場合，セグメントが 0.2 秒間隔でしか送信されなくなるため，極端にスループットが悪化する．

次に，TCP デッドロックの詳細について説明する．

2.1 送信バッファが MSS に比べて十分大きくない場合

TCP では，セグメントの喪失に備えて，確認応答されていないデータを送信バッファに保存しておかなければならない．そのため，送信バッファが最大セグメント長（MSS: Maximum Segment Size）の 3 倍より小さいと，**図 1** に示すようにデッドロックが発生する可能性がある[9]．このデッドロックは，データ転送が終了するまで繰り返し起こる可能性があり，スループットを著しく低下させる恐れがある．

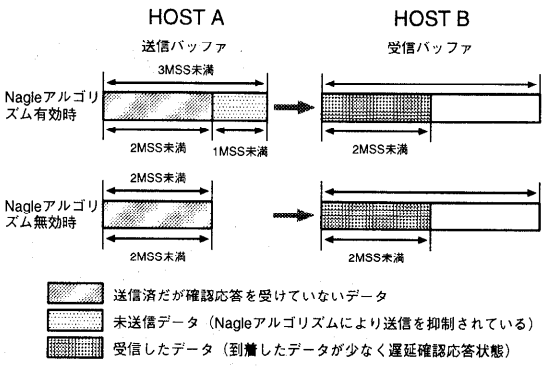


図 1 送信バッファが小さい場合のデッドロック
Fig. 1 Deadlock caused by smaller send buffer.

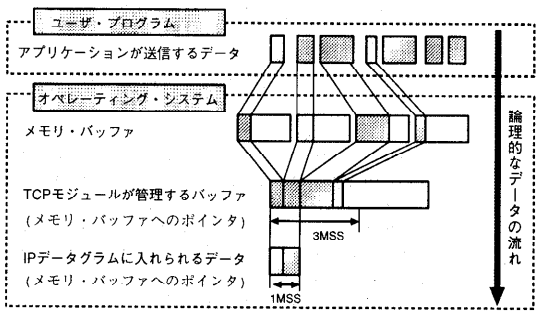


図 2 ネットワーク・モジュールの階層モデル
Fig. 2 Layer model of network module.

2.2 ネットワーク・モジュールの階層化の問題

アプリケーションが，システム・コールでデータを送信するときのメッセージの大きさによっては，TCP デッドロックが起きることがある．これは，ネットワーク・モジュールの階層化問題として知られている[10]．

多くの OS では，**図 2** に示すように，アプリケーション・プログラムと TCP モジュールの間に，TCP とアプリケーションを結ぶインタフェース☆が存在する．このインタフェースは OS の内部に含まれ，アプリケーションから OS へ渡される送信データをメモリ・バッファに格納したり，受信したデータをメモリ・バッファからアプリケーションへ渡す働きがある．メモリ・コピーによるオーバヘッドを避けるため，OS 内部の各層のモジュールは，メモリ・バッファに格納された同一のデータをポインタで参照する．多くの実装では，アプリケーションからシステム・コールによって OS へ渡されるデータは，データがまったく格納さ

☆ たとえば，BSD 系ではソケット，System V 系では STREAMS と呼ばれる．

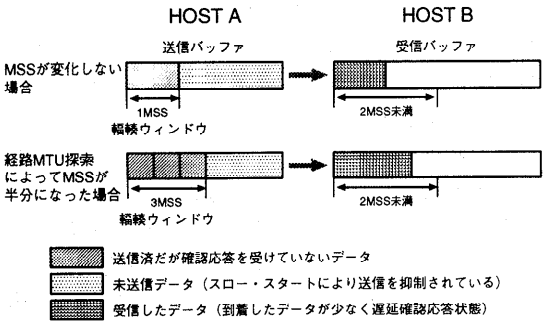


図3 スロー・スタートおよび，経路 MTU 探索によるデッドロック

Fig. 3 Deadlock caused by slow start and path MTU discovery.

れていない未使用の固定長バッファに格納される．使用されるメモリ・バッファの大きさに比べて，アプリケーションのメッセージ・サイズが小さい場合，メモリ・バッファの利用効率が悪くなる．メモリ・バッファの合計が3MSS以上あったとしても，合計3MSS未満のデータしか格納できないことが起こりうる．そうなると，2.1節と同じ状態になり，デッドロックが発生する．

また，このインタフェースが管理しているメモリ・バッファの空き領域が少なくなった場合，TCPへの送信処理が行われずに，バッファが大きくなるまで処理が中断されるように実装されている場合がある．これは一種のフロー制御といえるが，このフロー制御がTCPのフロー制御と独立に行われると，デッドロックが起きる可能性がある．

2.3 スロー・スタートの問題

送信ホストは，通信開始時にスロー・スタートの機構により，1MSSのセグメントを1つしか送信しない．しかし，受信ホストは，さらなるセグメントの到着を待ち，すぐに確認応答をしない．そのため，最初の1パケット目は**図3**の上段に示すようにデッドロックが発生する☆．

2.4 経路 MTU 探索の問題

経路 MTU 探索（Path MTU Discovery）[12]は，IPv4ではオプションの機能であり，最近まであまり実装されていなかった．しかし，IPv6では，ルータがMTU（Maximum Transmission Unit）を超えるデータグラムのフラグメント処理をしなくなるため，経路MTU探索の実装が強く求められるようになってきた．しかしながら，この経路MTU探索の機構は，TCPデッドロックを引き起こす新たな原因となる．

一般的なTCPの実装では，コネクションの確立時に，通信するホスト間で互いに最適だと判断されたMSSの値が交換される．そして小さい方の値が実際のデータ転送に使用されるMSSと決められる．このようにして決定されたMSSは，両ホストで同じ値になり，通常はコネクションが切断されるまで変化しない．

しかし，送信ホストに経路MTU探索が実装されている場合は，データ転送を開始したあとで正しい経路MTUの値が求まり，その値を基にMSSの値が再設定されることになる．そうなると，データ転送開始前に決定されたMSSの値は，経路MTUから求めたMSSの値とは必ずしも一致しない．現在のTCPの実装では，コネクション確立後にMSSの値が変更されても，その値は受信ホストには通知されない．また，仮に通知されたとしても受信時に無視される☆☆．そのため，受信ホストは，送信ホストのMSSの値とは無関係に，コネクションの確立時に交換されたMSSの値を使って遅延確認応答を制御することになる．

送信ホストのMSSが，コネクション確立時に決定された値よりも小さく変化する場合は，図3の下段に示すようなデッドロックが発生する可能性がある．大きく変化する場合は，遅延確認応答が行われなくなり，確認応答パケットが増加し，ネットワークやCPUの負荷が増加する可能性がある．

3. 既存の TCP デッドロック問題の解決案

今までに，TCPデッドロック問題の解決策がいくつか提案されている．それらの解決案と，その問題点について述べる．

☆ このデッドロックは，ウィンドウをよりゆっくりと拡大させるため，スロー・スタートによる混雑の回避に良い影響を与えるのではないかと考えられるかもしれない．しかし，TCPでは，混雑の回避は輻輳ウィンドウがスロー・スタート閾値を超えた後の，輻輳回避段階で行われるように設計されている[11]．そのため，最初の1パケットだけ遅延させても，輻輳回避段階へ到達する時間が遅くなるだけで，混雑の回避に良い影響を与えるわけではない．

☆☆ MSSの値が変更されるたびに，MSSオプションでMSSの値を伝えるようにTCPを変更したとしても，デッドロック問題を完全に解決することはできない．MSSを通知するためにはTCPのオプションを使わなければならず，送信するパケットにはその分のヘッダ・オーバヘッドが付く．そのため，MSSを通知するパケットでは，新しく求まった1MSS分のデータを運ぶことができず，結局，ウィンドウが2MSS以下の場合にデッドロックが起きることになる．ただし，すべてのTCPセグメントでMSSオプションを付けるか，MSSの値からMSSオプションの大きさをあらかじめ引いておけば問題を解決できる．しかし，MSSは頻繁に変更されるものではなく，ヘッダのオーバヘッドが大きくなることのほうが，より大きな問題になるといえる．

3.1 今までに提案されたTCPデッドロック問題の解決案

i. 送信バッファを大きくする解決案

Comerら[9]は，送信バッファをつねにMSSの3倍以上に設定すれば，2.1節の「送信バッファがMSSに比べて十分大きくない場合」のデッドロックを解決できると主張した．

ii. 遅延確認応答を止める解決案

Moldeklevら[13]は，遅延確認応答処理を止めることで，デッドロック問題を解決出来ると主張した．

iii. PUSHフラグで遅延確認応答を制御する解決案

NetBSD 1.1では，PUSHフラグが設定されたセグメントを受信した場合に，遅延なく確認応答するように実装されている☆．

3.2 今までに提案されたデッドロック解決案の問題点

iの解決案は，2.1節のデッドロック以外は解決できないという欠点がある．さらに，バッファ・サイズの大きさを規定すること自体に問題がある．既存のデータリンクの中には，IPv4の最大パケット長である64 koctetを超える巨大なMTUのネットワークが存在する．また，近い将来，腕時計やICカードなど，少量のメモリしか搭載できない超小型の機器がTCP/IPを利用して通信するようになる可能性がある．また，IPv6ではアドレス空間が莫大な数に増加するとともに，4 Goctetまでの巨大なMTUのデータリンクが利用可能になる．そのため，ありとあらゆる物をインターネットに接続しようと試みられるに違いない．MTUやバッファ・サイズを規定することは，ネットワークやホストの仕様を制限する特別な条件になりかねない．このことから，MTUやバッファ・サイズとは無関係にデッドロックの問題が解決することが望ましい．

iiの解決案では，すべてのデッドロック問題を解決できると期待できる．デッドロックの原因は，すべて遅延確認応答が関係しているため，確認応答を遅延させなければ，デッドロックは発生しなくなると考えられるからである．しかし，遅延確認応答を止めるという方法自体に問題がある．遅延確認応答は，シリー・ウィンドウ・シンドローム（SWS: Silly Window Syndrome）と呼ばれるネットワークの帯域の利用効率を悪化させる現象を回避するために提案された[7]．現在のインターネットでは多彩なデータリンクが用いられていることを考えると，SWSの防止は必須であり，SWS再発の可能性のあるこの解決案は受け入れられるものではない．

iiiの解決案は，PUSHフラグが設定されずに，デッドロックが起きる場合は解決できない．PUSHフラグとデッドロックの間には完全な相関関係はなく，デッドロックが起きるときにPUSHフラグが設定されるとは限らない．そのため，デッドロックの根本的な解決にはなっていない．

また，TCPにはトラヒックを軽減する仕組みが備えられているが，iiとiiiの解決案はその仕組みを動作させなくなるという問題がある．これは，iiとiiiの提案が，TCPのデータ転送モデルにおける遅延確認応答とPUSHフラグの役割をきちんと考慮していないためである．これらの提案の問題点を明白にするため，次の節では，TCPのデータ転送モデルと，遅延確認応答，PUSHフラグの関係について説明し，iiとiiiの実装がTCPのデータ転送に与える影響について説明する．

3.3 デッドロック解決策とTCPのデータ転送モデル

TCPは，バルク・データ転送と，インタラクティブ・データ転送という，2つのデータ転送モデルを扱うことができる[14]．また，即時性を必要とするインタラクティブ・データ転送も扱うことができる．

3.3.1 バルク・データ転送モデル

バルク・データ転送では，**図4**に示すように片方向に大量のデータが転送される．Nagleアルゴリズムにより1 MSS単位でデータが送信される．また，遅延確認応答により，2 MSSのデータを受信するたびに1つの確認応答パケットが送信される．

このモデルの場合，iiの実装では1 MSSごとに確認応答が行われることになり，確認応答パケットが通常の2倍の数に増加する．この増加は，ネットワークや

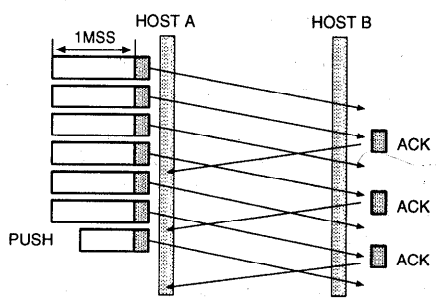


図4 バルク・データ転送
Fig. 4 Bulk data transfer.

☆ FreeBSD 2.1などオプションで同様の処理ができる実装がいくつか存在する．

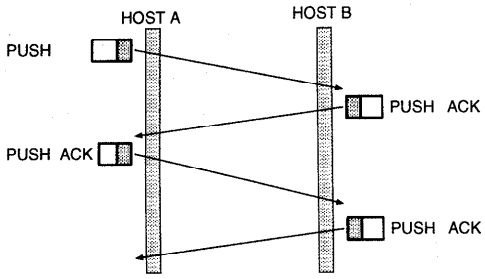


図5 インタラクティブ・データ転送
Fig. 5 Interactive data transfer.

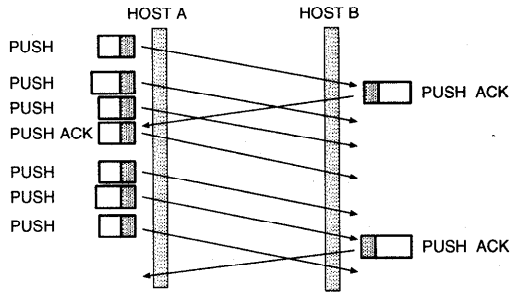


図6 リアルタイム性のあるインタラクティブ・データ転送
Fig. 6 Interactive data transfer with real-time.

CPU への負荷となり，スループットが低下する原因となる．

3.3.2 インタラクティブ・データ転送モデル

インタラクティブ・データ転送では，**図5** に示すように，アプリケーションはデータを送信したら，処理を停止して応答を待つ．このセグメントでは PUSH フラグが設定される☆．PUSH フラグは，そのセグメントの中に，アプリケーションに処理してほしいデータの単位が含まれていることを意味する．受信ホストでは，PUSH フラグが設定されたセグメントは，バッファリングされず，すぐにアプリケーションに送られる．アプリケーションは受信したデータを処理して，返事となるデータを作成し送信する．

インタラクティブ・データ転送では，データの送信と同じパケットで確認応答が行われる．これはピギーバックと呼ばる．TCP では，データの送信と確認応答を同一のヘッダで処理するため，このようなことができる[15]．ピギーバックが行われると，パケットの数が減るため，ネットワークの負荷や CPU の負荷を下げることができる．なお，遅延確認応答が行われなければピギーバックは起こらない．アプリケーションが受信したデータを処理し，返事を送信するまで，確認応答が遅延されなければ，ピギーバックは起きないのである．

なお，このデータ転送モデルでは，どんなに確認応答が遅れようとも TCP デッドロックは発生しない．なぜならば，データを送信すると，相手のホストからの返事を待つ状態になり，返事が来るまで次のデータを送信することはないからである．

このモデルの場合，ii や iii の実装では，ほとんどすべての受信セグメントに対して確認応答が行われることになる．このためピギーバックが行われなくなり，無駄な確認応答が送信されることになる．また，このデータ転送モデルの性質から，PUSH フラグが設定されたセグメントは，ピギーバックされる可能性が強いといえる．したがって，PUSH フラグが設定されたセグメントは遅延させるべきであり，iii の実装は根本的に間違っている．

3.4 即時性を必要とするインタラクティブ・データ転送モデル

ウィンドウ・システムや，機械制御など，即時性が求められるインタラクティブ・データ転送では，**図6** に示すようなデータ転送が行われる．このモデルは，TCP の Nagle アルゴリズムを無効にすることで実現される．図は，HOST A ではクライアント・プログラムが動作し，HOST B ではサーバ・プログラムが動作している場合である☆☆．

クライアントでイベントが発生するたびに，PUSH フラグが設定された小さなイベント・メッセージがサーバへ送信される．サーバはそのメッセージを受信してイベントの内容を調べる．クライアントへ返事を送信する必要がある場合はメッセージを送信するが，必要のない場合は送信しない．クライアントからサーバへ送信されるメッセージの確認応答は，理想的な場合にはピギーバックされる．また，サーバからクライアントへ送信されるメッセージの確認応答も，ピギーバックされる可能性がある．このように，TCP では，遅延確認応答処理により，応答の即時性を犠牲にすることなく，無駄な確認応答パケットを減らせるような設計になっている．

このモデルの場合，ii や iii の実装では，ほとんどすべてのセグメントに対して確認応答が行われることになる．その結果，無意味な確認応答パケットが大量に

☆ なお，データが MSS より大きい場合は，バルク・データ転送とまったく同じになり，データは MSS 単位に分割されて転送される．この場合，データの最後のセグメントでのみ PUSH フラグが設定される．

☆☆ X Window System の場合は用語が混乱するが，図の HOST A では X サーバが動作しており，HOST B では X クライアントが動作していると考える．

送信されることになり，ネットワークやCPUに過大な負荷となる危険性がある．

4. TCPデッドロック問題の解決策の提案

デッドロックが起きる根本的な原因は，受信ホストが送信ホストの状態を知ることなく，独立して遅延確認応答処理をしていることにある．送信ホストでは，送信バッファの残りサイズ，MSSの大きさ，送信処理が停止するかどうかなど，遅延確認応答処理をする上で必要な情報が変化する．それにもかかわらず，それらの情報は受信ホストには伝えられない．たとえば，経路MTU探索の問題の場合，MSSオプションを利用してつねにMSSの値を通知し続ければ，MSSが変化してもデッドロックを防ぐことができると予想される．しかし，このようにして個々の問題を別々に解決するのはあまり望ましいことではない．なぜならば，IPやTCPの機能に変更が加えられたときに，新たなデッドロック問題が発生する危険性があり，そのたびにTCPの仕様に修正を加えなければならなくなる可能性があるからである．そこで，本論文では，個々の問題によらずに，TCPデッドロック問題のすべてを，統一的な方法で解決することを目指す．

また，3.1節のiiやiiiの解決策は，TCPのデータ転送モデルを考慮していないために，デッドロックが起きないときでもトラヒックが増加するという問題がある．本論文では，3.3節で述べたTCPのトラヒックを軽減するための仕組みに影響することなく，デッドロック問題の解決を目指す．

以上のことを実現するため，本論文では，送信ホストが受信ホストの確認応答を制御することで，デッドロックを解決する方法を提案する．**図7**に示すように，TCPヘッダの予約ビット[15]に，NDAフラグ（Never Delay ACK flag）とFDAフラグ（Force Delay ACK flag）という2つの制御フラグを追加する．

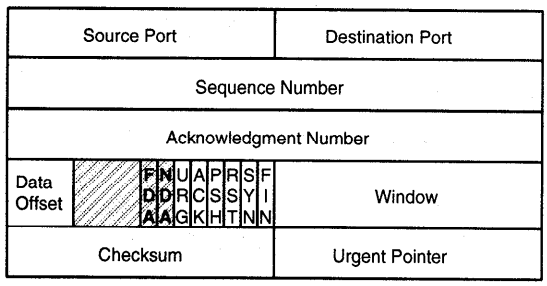


図7 提案するTCPのヘッダ
Fig. 7 The new TCP Header.

4.1 NDAフラグ（Never Delay ACK flag）

NDAフラグは遅延確認応答をせずに，すぐに確認応答を返送させるためのものである．このフラグを利用して，データを2MSS送信するたびに確認応答するように制御したり，デッドロックの発生を防いだりする．

送信ホスト

- 既存のTCPと挙動を同じにするため，2MSSのデータを送信するたびに1にする．
- デッドロックを回避するため，確認応答が来ない限り送信処理が停止する場合に1にする．

受信ホスト

- 1のときには，遅延なくそのセグメントに対する確認応答をする．

4.2 FDAフラグ（Force Delay ACK flag）

FDAフラグは，受信ホストがどんなにたくさんのセグメントを受信したとしても，すぐに確認応答をせず，遅延確認応答をするように制御するためのフラグである．このフラグにより，不必要な確認応答パケットの送信を防ぐ．ただし，確認応答が遅延しすぎることによるセグメントの再送を防ぐため，従来のTCPの仕様[8]と同様に，確認応答の遅延時間の上限は0.5秒とする．

送信ホスト

- データ・セグメントのすべてでFDAフラグを1にする．ただし，この場合は，NDAフラグで確認応答を正しく制御しなければならない．

受信ホスト

- 1のときには，そのデータ・セグメントに対する確認応答を遅延させる．ただし，従来のTCPの遅延確認応答と同様に，遅延時間の最大値は0.5秒以内でなければならない．
- NDAフラグが1のときはNDAフラグの処理が優先され，FDAフラグの値は無効となる．

なお，このFDAフラグは受信処理に対して状態を持たせないという役割を持つ．NDAフラグやFDAフラグを配置するTCPヘッダの予約ビットは，経路の途中にTCP/IPヘッダ圧縮機能[16]を使うネットワークがあると0にクリアされる．インターネットでは経路が変動する可能性があるため，FDAフラグが受信ホストまで伝わったり伝わらなかったりする可能性がある．このこのことが新たなデッドロックとなることを防ぐため，送信セグメントのFDAフラグをすべて1にする．これにより，予約ビットが0にクリアされる場合は従来のTCPと同じ遅延確認応答処理を行い，0にクリアされない場合はNDAフラグを基にした処

理を行うことになる．そのため，最悪の場合で従来と同じ性能を維持できる．

5. 実　　装

本提案の有効性を評価・検証するため，実装と実験を行った．インターネットのプロトコルでは，実装して動作させることが非常に重要視される．実装しなければ，実際に動作するのか，また，効果があるのか分からないからである．また，TCP デッドロックに関する研究[9),10),13)]は，特定の OS を対象にして原因が深く追究されてきたという歴史がある．このことからも，特定の OS 固有の問題を含んでいたとしても，実装して評価することは十分に意義のあることだといえる．

実装は BSDI 社の Internet Server V3.0（BSD/OS 3.0）をベースとして利用した．この OS は 4.4BSD Lite2 に基づいた OS であり，BSD の TCP/IP プロトコルスタックを基に，BSDI 社が改変したものが使われている．また，経路 MTU 探索が実装されているため，それが原因で起こるデッドロックに関する評価も行うことができる．多くの TCP/IP プロトコル・スタックは，BSD の実装を手本にして発展してきたため，本 OS への実装は他の OS の場合にも参考になると考えられる．

送信処理

送信処理では，(1) すべてのデータ・セグメントで FDA フラグを 1 にする，(2) データを 2 MSS 送信するたびに NDA フラグを 1 にする，(3) デッドロックの回避のために NDA フラグを 1 にするように実装した．(3) の処理は，データ送信を停止する直前の送信セグメントで行わなければならない．なお，セグメントの送信時に，次のセグメントの送信が停止するかどうか，分かる場合と，分からない場合がある．それぞれの実装について詳しく説明する．

まず，送信停止が分かる場合について述べる．送信可能なセグメントが 1 つあるとき，そのセグメントの送信後に次の状態になると，それ以降のセグメントの送信が停止する．

(1) 送信可能なウィンドウの大きさが，1 MSS 未満（Nagle アルゴリズム有効時）または，0（Nagle アルゴリズム無効時）になる場合．

(2) 送信ソケット・バッファの残りの大きさが，1 MSS 未満（Nagle アルゴリズム有効時）または，0（Nagle アルゴリズム無効時）になる場合．

これらの時に NDA フラグを設定する．ただし，NDA フラグを 1 に設定したセグメントがすでに送信されている場合は，確認応答パケットが来ることが期待される．そうなれば，デッドロックは発生しなくなると考えられる．不必要な確認応答が増加することを防ぐため，すでに NDA フラグを 1 に設定したセグメントが送信済みである場合は，NDA フラグを 0 に設定することにした．

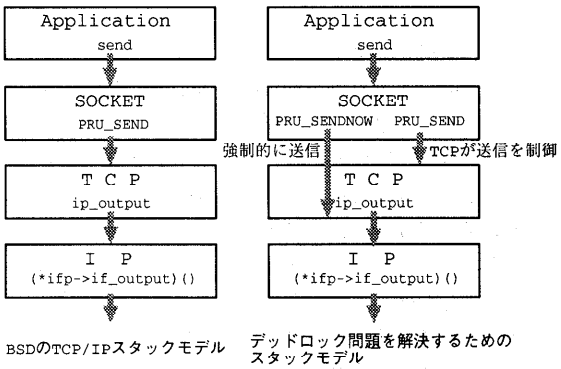


図 8　ソケットと TCP のモジュール間通信の強化
Fig. 8　New socket message.

次に，送信停止が分からない場合について述べる．2.2 節で述べた，ソケットのバッファが足りなくなり，処理が停止するタイミングは事前には分からない．アプリケーションのメッセージ・サイズによって，停止するかしないか変わるからである．そこで，ソケットが sbwait() を呼び出して処理を停止するときに，NDA フラグを 1 に設定してバッファに格納されている未送信データを強制的に送信するように実装した．この処理を実現するため，図 8 に示すように PRU_SENDNOW メッセージを追加した．ソケットは，ソケット・バッファが足りなくなり処理を停止するときに，TCP に PRU_SENDNOW メッセージを送る．このメッセージを TCP が受信した場合，NDA フラグを 1 に設定して，バッファに格納されている未送信データを強制的に送信する．

ただし，この場合も，送信停止が分かる場合と同様に，すでに NDA フラグを 1 に設定したセグメントが送信されている場合は，確認応答が来ることが期待される．確認応答が来れば，ソケットのバッファが解放され，より多くのデータを送信できるようになり，デッドロックが発生しない可能性がある．そこで，NDA フラグを 1 に設定したセグメントが送信中の場合は，PRU_SENDNOW メッセージは PRU_SEND メッセージと同じ意味に解釈され，強制的な送信処理は行われないように実装した．

受信処理時（確認応答処理時）

BSD/OS 3.0 の実装では，次の場合に確認応答が送

信される．

(1) 2 MSS 以上のデータを受信したとき

(2) 受信バッファ（*so_rcv.sb_hiwat*）の 50%以上のデータを受信したとき

(3) 0.2 秒間隔のファスト・タイマが実行されたときに，未送信の確認応答がある場合

本提案の実装では，不必要に確認応答が増加することを防ぐため，FDA フラグが 1 の場合は，次の場合にのみ確認応答を送信するように実装した．なお，FDA フラグが 0 のときは，上記の (1) や (2) のときも確認応答するように実装した．

(1) NDA フラグが 1 のとき

(2) 0.2 秒間隔のファスト・タイマが実行されたときに，未送信の確認応答がある場合

6. 検証実験

本論文の提案が，実際のネットワーク・システム上で，有効に働くかどうかを検証するために実験を行った．本提案の効果の程度を比較するために，他の 3 つのアルゴリズムでも同じ実験を行った．使用したアルゴリズムは次の 4 種類である．

実装 1　BSD/OS 3.0 の実装のままで無変更
実装 2　つねに確認応答を遅延させない
実装 3　PUSH フラグが立っているときは確認応答を遅延させない
実装 4　本論文の提案

検証実験では，デッドロックが起こりうる環境で，バルク・データ転送のデータ・トラヒックを転送し，スループットや受信データ量の時間推移を測定した．また，カーネル内部の統計情報から，送信セグメントや確認応答の数を求めた．このとき，測定しているデータ以外のトラヒック（測定ツールの制御トラヒックなど）が，含まれないように処理した．測定は，送受信の両方のホストを同じ実装にして，その実装ごとに行った．なお，測定結果は 1 回の実験結果の値である．ただし，示した結果以外にも，予備実験を複数回行っており，再現性に関しては確認済みである．また，予備実験との比較から，スループットの精度は 0.1 Mbps 程度以上，測定時間の精度は 0.01 秒程度以上はあると思われる．

実験環境は，**図 9** または，**図 10** である．実験時の設定は図または表，本文中に記載されている場合を除き**表 1** に示したとおりである．

なお，BSD の実装では，送信バッファと受信バッファのうちの，大きさが小さい方の値がウィンドウの最大値を決める．このため，この節ではバッファ・サイズとウィンドウ・サイズという用語を特に区別せずに使用する．

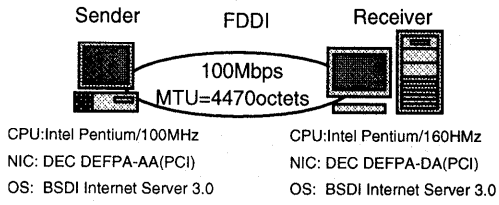


図 9 実験環境
Fig. 9 Environment of experiments.

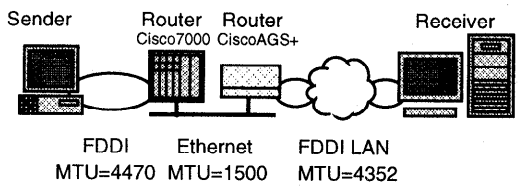


図 10 経路 MTU 探索の実験環境
Fig. 10 Environment of the path MTU discovery experiment.

表 1 実験のパラメータ（図，表または本文中に記載されている場合を除く）
Table 1 Parameter of experiments (ignored if it is described in the figure, table or text).

OS	BSDI Internet Server 3.0 (BSD/OS 3.0)（K300-001 patch apply）
メッセージ・サイズ	8192 byte 固定または，2048 byte 固定
バッファ・サイズ	65535 byte
MTU	4470 octet（*BSD/OS 3.0 のデフォルト）
MSS	4352 octet（*BSD/OS 3.0 のデフォルト）
ヘッダの長さ	IP:20 octet, TCP: 20 + 12 octet (TCP タイムスタンプオプション付き)
Nagle アルゴリズム	有効
測定ツール	Netperf☆（6.1 節，6.2 節，6.5 節） DBS☆☆（6.3 節，6.4 節）

6.1 実験 1：送信バッファが MSS に比べて十分大きくない場合

図 9 に示す環境で，バッファ・サイズを変えてスループットを計測した．結果を**表 2**〜**5** に示す．なおメッセージ・サイズは 8192 byte で，データ転送時間は 10 秒間である．

表 2 で，送信バッファ・サイズが 3 MSS 未満の領域，すなわち 12888 byte 以下の領域では，スループットが 1 Mbps 未満になっている部分がある．その領域で

☆ http://www.cup.hp.com/netperf/NetperfPage.html から入手できる

☆☆ http://www.ai3.net/products/dbs/から入手できる

表2 実装1：TCP バッファ・サイズとスループット（Mbps）
Table 2 Implement.1：TCP buffer sizes and mean throughput (Mbps).

送信バッファ（byte）	受信バッファ（byte） 4096	8192	12288	16384	32768	65535
4096	28.34	0.01	0.01	0.01	0.01	0.01
8192	24.81	14.53	12.99	0.33	0.33	0.33
12288	24.39	23.85	22.68	0.34	0.34	0.34
16384	24.22	35.94	41.32	37.94	38.73	38.54
32768	24.41	36.05	42.81	55.79	55.26	54.74
65535	24.09	36.12	42.61	55.54	54.80	54.56

表3 実装2：TCP バッファ・サイズとスループット（Mbps）
Table 3 Implement.2: TCP buffer sizes and mean throughput (Mbps).

送信バッファ（byte）	受信バッファ（byte） 4096	8192	12288	16384	32768	65535
4096	28.51	0.02	0.02	0.02	0.02	0.02
8192	24.63	33.33	32.52	33.54	33.22	33.57
12288	24.64	37.92	37.61	37.72	37.18	37.86
16384	24.35	43.54	47.60	48.75	48.22	47.61
32768	24.70	42.56	51.51	52.17	51.22	51.69
65535	24.39	42.06	50.99	50.69	51.11	49.72

表4 実装3：TCP バッファ・サイズとスループット（Mbps）
Table 4 Implement.3: TCP buffer sizes and mean throughput (Mbps).

送信バッファ（byte）	受信バッファ（byte） 4096	8192	12288	16384	32768	65535
4096	28.02	0.02	0.02	0.02	0.02	0.02
8192	24.60	21.05	25.28	23.05	22.42	19.56
12288	24.46	25.59	25.84	24.56	21.04	20.12
16384	24.49	36.34	41.05	37.19	37.12	38.45
32768	24.51	36.82	42.81	55.88	56.42	55.46
65535	24.34	37.01	42.19	56.19	55.21	56.17

表5 実装4：TCP バッファ・サイズとスループット（Mbps）
Table 5 Implement.4：TCP buffer sizes and mean throughput (Mbps).

送信バッファ（byte）	受信バッファ（byte） 4096	8192	12288	16384	32768	65535
4096	18.87	20.98	21.06	20.92	20.86	21.17
8192	24.27	28.39	30.66	29.42	28.91	30.52
12288	24.42	19.76	33.78	30.71	30.88	33.43
16384	24.07	35.95	41.48	41.17	41.92	41.30
32768	24.31	29.65	41.92	56.94	55.67	55.49
65535	24.01	35.06	41.89	54.41	54.17	56.35

はデッドロックが発生している．なお，その領域でも，受信バッファが小さい場合にはデッドロックが起きていない．BSD/OS 3.0 では，受信バッファの 50%のデータを受信したときに確認応答するため，受信バッファが小さければ 2 MSS 受信しなくても確認応答されるからである．

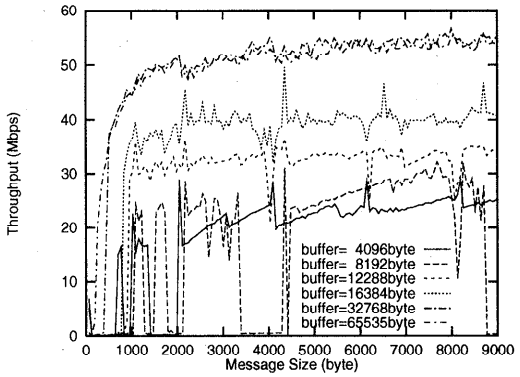


図11 実装1：メッセージ・サイズとスループット
Fig. 11 Implement.1: Message sizes and throughput.

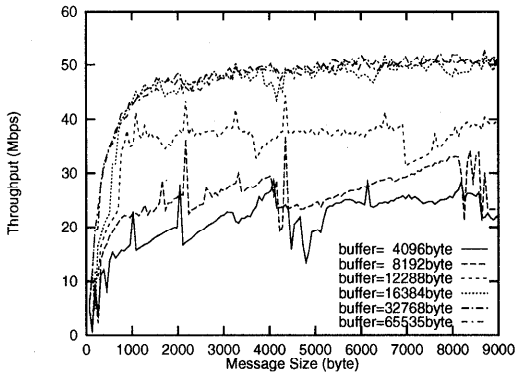


図12 実装2：メッセージ・サイズとスループット
Fig. 12 Implement.2: Message sizes and throughput.

実装4以外では，デッドロックが起きている．デッドロックが起きないはずの実装2でもスループットが小さい領域がある．これは，BSD/OS 3.0 では送信バッファがメッセージ・サイズに比べて極端に小さい場合は，すべての確認応答が来ていたとしても，すぐには送信処理が行われないような実装になっているからである．

6.2 実験2：ネットワーク・モジュールの階層化の問題

図9に示す環境で，メッセージ・サイズを 64 byte から 64 byte 単位で，約 9000 byte まで設定し，データ転送を 10 秒間行ったときのスループットを測定した．なお，各実装ごとに 6 種類のバッファ・サイズで測定した．結果を図 **11**～**14** に示す．

実装1は計測したすべてのバッファ・サイズで，実装3はバッファ・サイズが 8192 byte のときに，メッセージ・サイズによっては，スループットが極端に小さくなる場合があることが分かる．これは，2.2 節で述べたデッドロックが起きているためである．なお，すべての実装で，メッセージ・サイズが 0 byte に近く

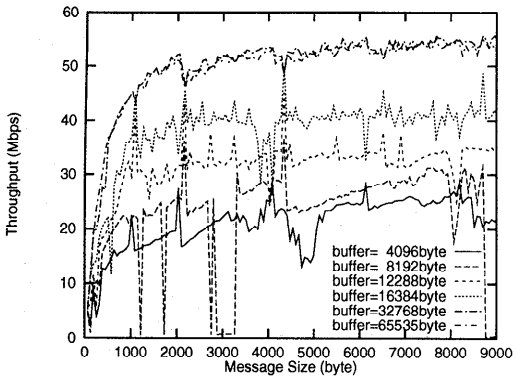


図13 実装3：メッセージ・サイズとスループット
Fig. 13 Implement.3: Message sizes and throughput.

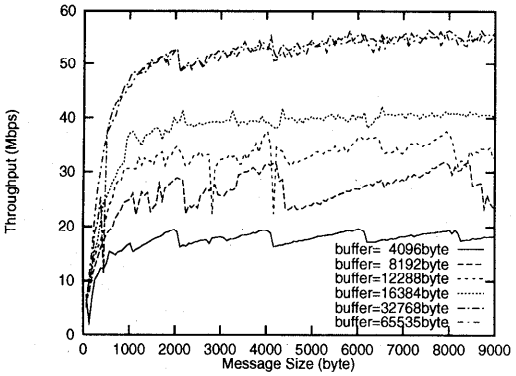


図14 実装4：メッセージ・サイズとスループット
Fig. 14 Implement.4: Message sizes and throughput.

なるに従って，スループットが小さくなる傾向がみられる．これはシステム・コールのオーバヘッドによるものであり，デッドロックが起きているわけではない．

6.3 実験3：スロー・スタートの問題

図9に示す環境で，メッセージ・サイズ2048 byteにしてデータ転送を行い，データ送信開始直後の受信データ量の時間推移を測定した．結果を**図15**に示す．この図の約0.0秒のときにコネクション確立のためのSYNパケットが送信されている．図をみると，実装1では0.15秒近くまでデータ転送が停止しており，デッドロックが発生していることが分かる．デッドロックの期間は，実際にはデータ送信時のタイミングにより0.0秒から0.2秒の間の値になる．デッドロックが発生する場合は，発生しない場合に比べて，平均約0.1秒の遅延時間が加えられることになる．高速なLAN環境では，ラウンド・トリップ時間は数ミリ秒であるため，このデッドロックによる約0.1秒の遅延は大きな遅延といえる．この遅延時間はLANの性能に関係なく加えられるため，人がインタラクティブ的に操作するアプリケーションなどの応答性の限界を，TCPが

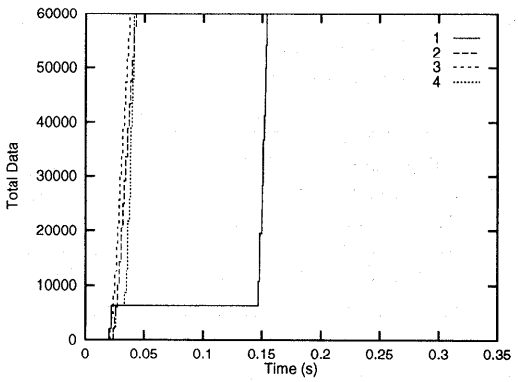


図15 各実装における通信開始時の受信データの時間推移
Fig. 15 Behaver of slow start.

決めることになりかねない．

なお，BSD/OS 3.0の実装では，SYNパケットに対する確認応答も輻輳ウィンドウを増加させる．そのため，通常のスロー・スタートと異なり，輻輳ウィンドウは2 MSSから始まる．このように実装すれば，デッドロックが起こらなくなるように思われるかもしれないが，実際にはデッドロックが発生する．

データの送信開始時に，アプリケーションが1 MSS未満のメッセージ・サイズでデータを送信すると，Nagleアルゴリズムの送信条件を満たすため，データはそのメッセージ・サイズのまま送信される．次のデータ以降は送信中のデータがあるため，Nagleアルゴリズムにより，セグメントは1 MSS単位で送信される．輻輳ウィンドウが2 MSSの場合は，最初に1 MSS未満のセグメントと1 MSSのセグメントの2つパケットが送信される．しかし，その結果，合計2 MSS未満のデータしか受信ホストに届けられず，デッドロックが発生することになる．

BSD/OS 3.0のスロー・スタートの実装でも，アプリケーションのメッセージ・サイズをMSSよりも大きくすればデッドロックは発生しない．しかし，アプリケーションはMSSを気にせずに作成されることが普通である．MSSの値はデータリンクのMTUに深く関係するため，結果として，データリンクのMTUを考慮してアプリケーションを作製しなければならなくなる．これは，ネットワーク・プロトコルの階層化モデルを考慮すると望ましいことではない．以上のことから，スロー・スタート開始時に，ウィンドウを2 MSSから開始してもデッドロックの解決策になっていないことは明らかである．

6.4 実験4：経路MTU探索の問題

図9に示す環境でデータ転送を行い，データ送信開始直後の受信データ量の時間推移を測定した．結果を

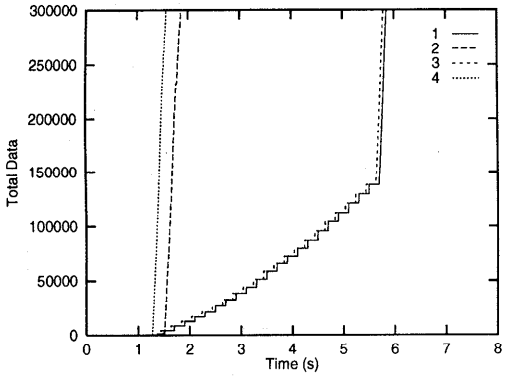


図 16 経路 MTU 探索に関する実験
Fig. 16 Experiment of the path MTU discovery.

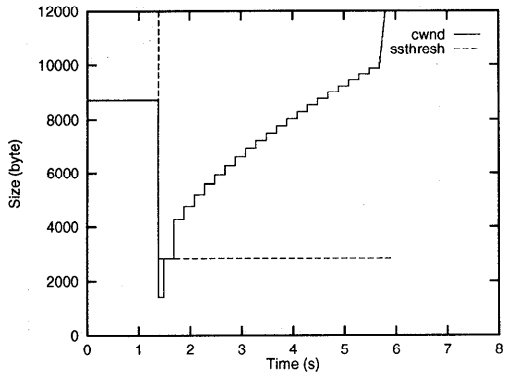


図 17 輻輳ウィンドウとスロースタート閾値の変化
Fig. 17 Congestion window and slow start threshold.

図 16 に示す．すべての実装で最初の 1 秒間はデータの転送が行われていない．また，実装 1 と実装 3 は 1～6 秒までデータが間欠的にしか転送されていない．

コネクションの確立時には，FDDI の MTU を基にした MSS の値がホスト間で交換される．しかし，最初のデータ・パケットは Ethernet の MTU よりも大きいため，ルータを通過できない．ルータから ICMP 到達不能メッセージ（要フラグメント）で Ethernet の MTU を通達され，次の送信以降はその MTU を基にした MSS で送信が行われる．BSD/OS 3.0 の実装では，ICMP によって経路 MTU が変更されても，TCP モジュールには通知されない．そのため，TCP モジュールはタイム・アウトによる再送で，はじめて MTU が変更されていることを知る．最初の約 1 秒間のデータ転送の停止は，この TCP の再送待ちの時間である．

実装 1 と実装 3 は 1～6 秒までデータが間欠的にしか転送されていない．カーネル内部の TCP トレース機能を利用して，TCP の輻輳ウィンドウとスロー・スタート閾値について調査した．結果を図 17 に示す．

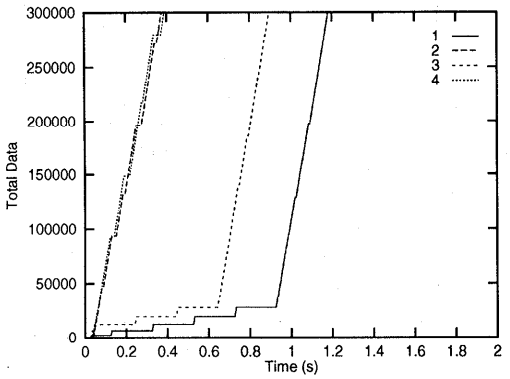


図 18 経路 MTU 探索に関する実験（経路 MTU がキャッシュされている場合）
Fig. 18 Experiment of the path MTU discovery when path MTU is cashed.

通常の TCP では，タイム・アウトの原因はすべて輻輳だと考えて実装されている．BSD/OS 3.0 では，経路 MTU 探索によるセグメントの喪失も特別扱いにせず，輻輳時と同じ処理が行われるように実装されている．そのため，輻輳ウィンドウの半分の値にスロー・スタート閾値が設定され，新しい MSS を基にスロー・スタートが行われる．輻輳ウィンドウが閾値を超えると，1 つの確認応答につき，(MSS^2/輻輳ウィンドウ) ずつしか増加しない．デッドロックが解消されるためには，コネクション確立時に決められた MSS の 2 倍になるまで，輻輳ウィンドウが拡大されなければならない．このため，比較的長い期間，繰り返しデッドロックが発生している．

なお，経路 MTU はある期間キャッシュされる．キャッシュされているときに，データを送信する実験も行った．結果を図 18 に示す．1 秒以内ではあるが，実装 1 と実装 3 はデータ転送開始時に間欠的なデータ転送を行っている．これは，コネクションの確立時に，キャッシュされている MTU を基にした MSS ではなく，FDDI の MTU を基にした MSS の値を交換していることが原因である．キャッシュされている MTU から求めた MSS を交換すれば，このデッドロックは起きず，そういう実装も存在する．しかし，これにはまだ議論の余地がある．現在，経路 MTU 探索をする場合に，MSS オプションでいくつの値を交換すべきかについて，TCP の標準は決められていない．そのため，各実装は独自の判断で MSS の値を決定している．しかし，IPv6 が本格的に運用されるまでに標準化されないと大きな問題を引き起こす可能性がある．

6.5 トラヒック量の評価

トラヒック量の評価をするため，図 9 の環境で，バッ

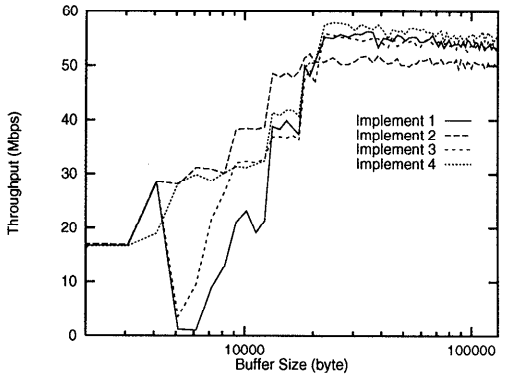


図 19 バッファ・サイズとスループット（Nagle 有効）
Fig. 19 Buffer size and mean throughput (Nagle ON).

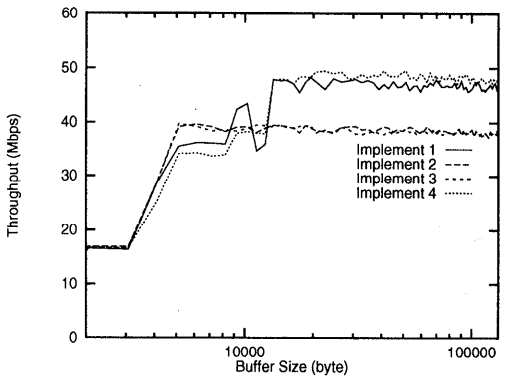


図 22 バッファ・サイズとスループット（Nagle 無効）
Fig. 22 Buffer size and mean throughput (Nagle OFF).

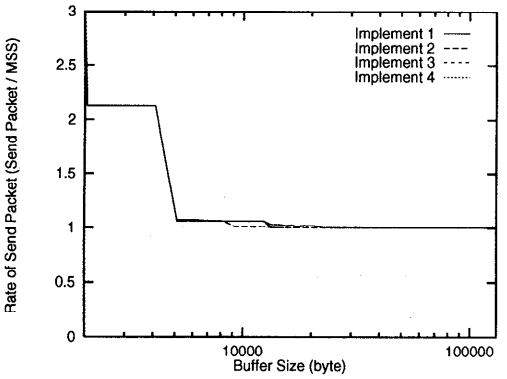


図 20 バッファ・サイズと送信データ 1 MSS あたりの送信パケット数（Nagle 有効）
Fig. 20 Buffer size and send packet par 1 MSS (Nagle ON).

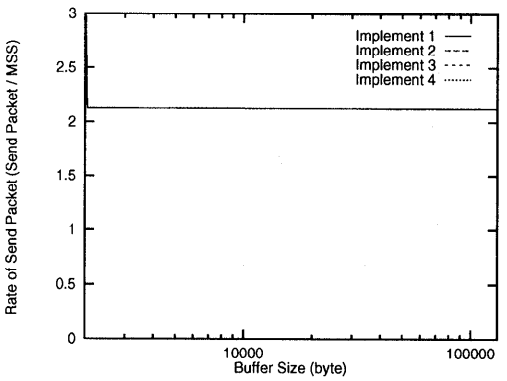


図 23 バッファ・サイズと送信データ 1 MSS あたりの送信パケット数（Nagle 無効）
Fig. 23 Buffer size and send packet par 1 MSS (Nagle OFF).

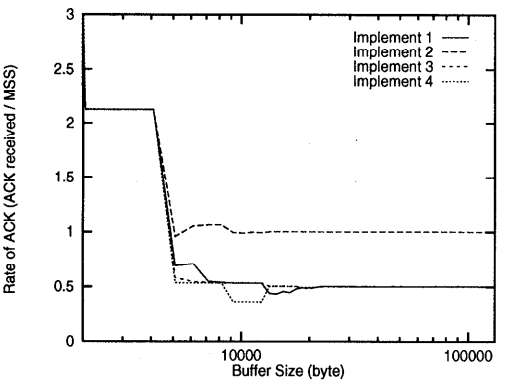


図 21 バッファ・サイズと送信データ 1 MSS あたりの確認応答パケット数（Nagle 有効）
Fig. 21 Buffer size and ACK packets par 1 MSS (Nagle ON).

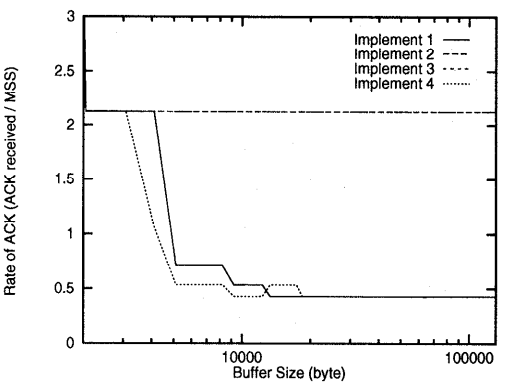


図 24 バッファ・サイズと送信データ 1 MSS あたりの確認応答パケット数（Nagle 無効）
Fig. 24 Buffer size and ACK packets par 1 MSS (Nagle OFF).

ファ・サイズごとのスループットと，送信データ 1 MSS あたりの送信パケット数，送信データ 1 MSS あたりの確認応答数を計測した．Nagle アルゴリズムが有効な場合の結果を図 **19**～**21** に，Nagle アルゴリズムを無効にした場合の結果を図 **22**～**24** に示す．

実装 1 と実装 4 は，Nagle アルゴリズムが有効な場合でも無効な場合でも，ほとんど同じトラヒック量である．しかし，実装 2 では，確認応答のトラヒック量

が，Nagle アルゴリズムが有効な場合は約 2 倍，無効な場合には約 5 倍になっている．実装 3 では，Nagle アルゴリズムが有効な場合はほとんど同じであるが，Nagle アルゴリズムが無効な場合には確認応答の割合が約 5 倍になっている．

Nagle アルゴリズムが有効な場合，実装 2 は，バッファ・サイズが 8〜18 kbyte という比較的小さな値のときには，他の実装よりスループットが約 20%高い．しかし，バッファ・サイズが 20 kbyte 以上の大きな値になると，逆にスループットが約 10%低くなっている．バッファ・サイズが 3 MSS や 4 MSS 未満の場合，1 MSS のデータを最大 2 つか 3 つしか送信できない．そのため，他の実装では 1 RTT あたり 1 つの確認応答しか行われないのに対し，実装 2 では 2 つ以上の確認応答が行われる．このため，実装 2 ではパイプライン的な効果が表れ，バッファが小さい場合にスループットが向上する．しかし，バッファが大きくなると，過剰な確認応答によってネットワークやホストの負荷が上昇し，スループットが低下する．

7. 検証実験のまとめ

本実験はインターネット上で発生しうるすべての状態を網羅しているわけではない．しかし，TCP デッドロック問題を解決しているかどうかを判断するには，十分に網羅できていると考える．本論文の提案である実装 4 は，デッドロックの原因として知られている 4 つの問題を解決できることがほぼ明らかである．また，実装 4 は現状の TCP の実装と比べ，トラヒックの増加がほとんどないことも明らかである．

実装 2 は多くの TCP デッドロック問題を解決できるが，トラヒックが増加するという問題がある．とくに，広帯域なネットワークの能力を使い切るためには，大きなバッファ（ウィンドウ）が必要になる．しかし，実装 2 ではバッファ（ウィンドウ）が大きくなると性能が悪化する．そのため，ネットワークの未来を考えると受け入れられない実装である．

実装 3 はデッドロックを部分的にしか解決できず，また，トラヒックが増加するため意味のない実装である．この実装は，TCP のデータ転送モデルをきちんと理解していない人によって行われたと考えられる．

以上から，本論文で提案する TCP の短期的なデッドロック問題の解決策は，デッドロックの防止に効果があり，十分に実用的であり，副作用も特にないと結論づける．

また，今回の実装では，データを 2 MSS 送信するたびに確認応答するように実装した．しかし，実装 2 のスループット特性から，確認応答の割合を動的に変化させることにより，スループットを向上できると見込まれる．制御が複雑になる可能性があるが，十分に検討する価値のある課題だと思われる．

8. 本提案のインターネットへの適用について

広く運用されているインターネットで，TCP のヘッダのフォーマットを変更することは，互換性上，問題となる可能性がある．しかし，本提案の場合は大きな問題とはならない．TCP の予約ビットは，送信時に 0 に設定され，また受信時には無視されなければならないことになっている．本論文で提案する NDA フラグと FDA フラグを実装しているホストと，実装していないホストが通信を行った場合は，デッドロック問題は解決しないが，現状の TCP の通信とまったく同じように通信できる．問題があるとすれば次の 3 点が考えられる．

(1) TCP/IP のヘッダ圧縮機能[16]を利用するネットワークを通る場合，NDA フラグ，FDA フラグがともに 0 に設定される．
(2) TCP の予約ビットが 1 になっていると誤動作するホストまたはルータがある場合．
(3) NDA フラグと FDA フラグに対応するフラグのいずれか，または両方に 1 をつけて送信するホストがある場合．

(1) で，予約ビットが 0 に設定される場合は，既存の遅延確認応答と同じ処理が行われる．その結果，デッドロック問題は解決しないが，現状の TCP とまったく同様に通信ができる．また，(2) や (3) の機器はほとんどないと考えられる．

IPv6 の場合はまだ実験段階であり，本格的に運用される前にこの提案が標準となれば，何の問題もなく TCP デッドロック問題を解決することができる．また，IPv6 では経路 MTU 探索が頻繁に行われるようになり，デッドロックの問題が深刻化する可能性がある．本提案はその問題を解決するための有効な手段となるため，本格的に運用される前に標準化する必要がある．

9. おわりに

本論文では，インターネットのトランスポート・プロトコルである，TCP の短期的なデッドロックの問題に焦点を当てて，その解決法について議論した．デッドロックを解決する方法として，送信ホストが遅延確認応答の処理を完全に制御する方法を提案した．また，TCP の遅延確認応答やピギーバック機構，PUSH 機

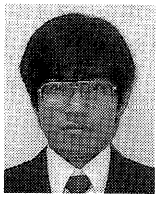
構に悪影響を与えることがないよう，TCP の転送モデルを考慮して，デッドロックを解消するように設計した．実装・実験によって，本論文の提案する解決策は，トラヒックの増加なしに，デッドロック問題を解決できることを示した．

今後，提案内容を IETF で議論を行い，Internet 標準となるように活動を展開する計画である．TCP のヘッダの予約ビットを利用するが，IPv6 の場合は運用される前に標準化すればまったく問題はなく，早急な標準化活動が必要である．

参考文献

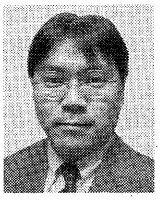
1) Deering, S. and Hinden, R.: Internet Protocol, Version 6 (IPv6) Specification, RFC 1883 (1996).
2) Huitema, C.: *IPv6: The New Internet Protocol*, Prentice Hall PTR (1996).
3) Jacobson, V., Braden, R. and Borman, D.: TCP Extensions for High Performance, RFC 1323 (1992).
4) Mathis, M., Mahdavi, J. and Floyd, S.: TCP Selective Acknowledgment Options, RFC 2018 (1996).
5) Nagle, J.: Congestion Control in IP/TCP Internetworks, RFC 896 (1984).
6) Stevens, W.: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, RFC 2001 (1997).
7) Clark, D.D.: Window and Acknowledgment Strategy in TCP, RFC 813 (1982).
8) Braden, R.: Requirements for Internet Hosts – Communication Layers, RFC 1122 (1989).
9) Comer, D.E. and Lin, J.C.: TCP Buffering And Performance Over An ATM Network, Purdue Technical Report, CSD-TR 94-26 (1994).
10) Crowcroft, J., Wakeman, I., Wang, Z. and Sirovica, D.: Is Layering Harmful?, *IEEE Network Magazine*, Vol.6, pp.20–24 (1992).
11) Jacobson, V.: Congestion Avoidance and Control, *Proc. ACM SIGCOMM '88, Stanford, CA* (1988).
12) Mogul, J. and Deering, S.: Path MTU Discovery, RFC 1191 (1990).
13) Moldeklev, K. and Gunningberg, P.: How a Large ATM MTU Causes Deadlocks in TCP Data Transfers, *IEEE/ACM Trans. Networking*, Vol.3, No.4, pp.409–422 (1995).
14) Stevens, W.R.: *TCP/IP Illustrated*, Volume 1, *The Protocols*, Addison-Wesley (1994).
15) Postel, J.: Transmission Control Protocol, RFC 793 (1981).
16) Jacobson, V.: Compressing TCP/IP Headers for Low-Speed Serial Links, RFC 1144 (1990).

(平成 9 年 5 月 19 日受付)
(平成 9 年 10 月 1 日採録)



村山 公保（学生会員）

平成 4 年東京学芸大学教育学部特別教科教員養成課程理科専攻地学専修卒業．同年日本電気技術情報システム開発（株）入社．平成 6 年同退社．同年奈良先端科学技術大学院大学情報科学研究科博士前期課程入学．平成 8 年同修了．現在同博士後期課程在学中．インターネットアーキテクチャ，ネットワーク性能評価に関する研究に従事．日本ソフトウェア科学会，電子情報通信学会各会員．



山口 英（正会員）

昭和 61 年大阪大学基礎工学部情報工学科卒業．同大学大学院前期課程修了．同後期課程を退学し，同大学情報処理教育センター助手に着任．平成 4 年奈良先端科学技術大学院大学情報科学センター助手，同助教授を経て，平成 5 年同大学情報科学研究科助教授．工学博士．インターネットアーキテクチャ，コンピュータセキュリティの研究に従事．